Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Sulautetut ohjelmistot

2015

Aku Hosio

DYNAAMISEN LAITEHALLINTAKIRJASTON KEHITTÄMINEN CATVISOR COMMANDERILLE

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut ohjelmistot

2015 | 31

Juha Nikkanen, Marko Vilola

Aku Hosio

DYNAAMISEN LAITEHALLINTAKIRJASTON KEHITTÄMINEN CATVISOR COMMANDERILLE

Opinnäytetyön tarkoituksena oli kehittää dynaaminen laitehallintakirjasto Telesten CATVisor Commander -ohjelmistolle ja Androidille sekä määritellä näille yhteiset käyttöliittymämääritykset. Dynaamisen käyttöliittymäkirjaston tulee myös tukea EMS 5 -viestintäprotokollan mukaisia viestimääritelmiä.

Telesten laitehallintakirjastojen käyttöliittymissä ja toiminnoissa on paljon yhtäläisyyksiä. Vain säätimien määrä vaihtelee laitteesta riippuen. Työssä kartoitettiin näitä yhtäläisyyksiä ja niitä käytettiin hyväksi käyttöliittymämäärittelyjen suunnittelussa.

Osissa Telesten laitehallintakirjastoissa käytetään muista laitehallintakirjastoista poikkeavia käyttöliittymäominaisuuksia ja toiminnallisuuksia. Työssä pohdittiin tällaisten ominaisuuksien toteuttamisen hankaluutta.

Työn tuloksena on kehitetty dynaamisen laitehallintakirjaston prototyyppi, joka pystyy jäsentämään laitteen viestimääritelmät ja käyttöliittymämääritelmät.

ASIASANAT:

Dynaamisuus, generointi, laitehallinta, käyttöliittymä

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded Software

2015 | 31

Juha Nikkanen, Marko Vilola

Aku Hosio

DEVELOPING A DYNAMIC DEVICE CONTROL LIBRARY FOR CATVISOR COMMANDER

The goal of the thesis was to develop a dynamic device control library for the Teleste CATVisor Commander -software and Android, and to define common UI definitions. The dynamic user interface library must also support message definitions in accordance with the EMS 5 messaging protocol.

The UI and functionality of the Teleste user interface libraries have many features in common. Only the number of controls changes depending on the device. In this thesis the common features were pointed out and used to define the necessary UI definitions for controls.

Only a small number of Teleste's user interface libraries use specific UI features and functionality. In the thesis there is also discussion on the problems of implementing these complex features.

The result of thesis is a prototype of a dynamic device control user interface library for both Windows and Android, which can parse common device UI and message definitions.

KEYWORDS:

Dynamic, generic, device control, user interface

SISÄLTÖ

LIITTEET

KUVAT

TAULUKOT

KÄYTETYT LYHENTEET

ATL	Active Template Library, Microsoftin kehittämä kirjasto, jonka tarkoituksena on yksinkertaistaa COM sovellusten kehittämistä. ATL tarjoaa erilaisia makroja ja funktioita useasti tarvittaviin toimintoihin
COM	Component Object Model, Microsoftin kehittämä rajapinta teknologia, jonka avulla eri sovelluskomponentit voivat kommunikoida keskenään esimerkiksi eri ympäristöistä tai koneista
C++	Ohjelmointi kieli, joka tarjoaa korkeatasosoisista ohjelmointikielistä geneerisiä ominaisuuksia sekä mahdollistaa olio-ohjelmoinnin ja samaan aikaan mahdollistaa alhaisentason muistihallinnan
HFC	Hybrid Fibre-Coaxial, telealalla käytetty termi, jolla tarkoitetaan laajakaistaverkkoa, joka kostuu optisesta kuidusta ja koaksiaalikaapelista
Java	Oraclen kehittämä korkeatason ohjelmointikieli
JSON	JavaScript Object Notation, avoimen standardin tiedonvälitysmuoto, joka pohjautuu JavaScript komentosarjakieleen
TSEMP	Teleste Simple Element Management Protocol, Telesten laitteiden viestintäprotokollan nimi
Viewer	Katseluohjelma, ohjelmisto tai ohjelmistokomponentti, joka näyttää käyttäjälle visuaalisen toteutuksen sen käsittelemästä tiedosta
WTL	Windows Template Library, Microsoftin kehittämä epävirallinen kirjasto Windows sovellusten kehittämistä varten, joka pohjautuu ATL:n ja tarjoaa MFC:n kaltaiset käyttöliittymä kehitysmahdollisuudet
XML	Extensible Markup Language, W3C kehittämä tiedonvälitysmuoto
EMS	Element Management System, Telesten käyttämä yleisnimitys Telesten laitteiden kommunikaatio teknologioista

1 JOHDANTO

Työn tarkoitus on selvittää Teleste Commander laitehallintakirjastojen käyttöliittymien dynaamisen generoinnin mahdollisuudet. Tavoitteena on kehittää prototyyppi toimivasta dynaamisesta laitehallintakirjastosta sekä määritellä ja kuvata tätä varten tarvittavat tietorakenteet.

Commanderin sekä Windows- että Android-versioiden tulee käyttää yhteisiä viesti- ja käyttöliittymämääritelmiä, mutta alustoilla käytetään eri ohjelmointikieliä, joten tarvitaan myös kaksi toteutusta. Tietorakenteen jäsentimet tulevat eroamaan toisistaan, tiedonkäsittelyssä logiikoiden yhtäläisyys on silti toivottua.

Molempien alustojen tulee tukea Teleste EMS 5 -viestintäprotokollaa. Protokollaa tukeva kirjasto on jo käytössä Telesten nykyisessä Android-sovelluksessa, joten kirjaston toteutuksesta voidaan ottaa mallia ja hyödyntää Windows-puolen toteutusta tehtäessä.

Aluksi kuvataan lyhyesti Teleste CATVisor -ohjelmistoperheen nykytilannetta. Luvussa 3 esitellään lyhyesti Telesten EMS -viestintäprotokolla. Luvussa 4 ja 5 käydään läpi työn toteutuksen kannalta tärkeitä tekniikoita ja mitä lisäyksiä ja muutoksia ohjelmistoon piti tehdä dynaamisuuden tukemiseksi. Luvussa 6 käydään läpi työn toteutusta: tietorakenteen suunnittelua, tietorakenteen jäsentämistä, sekä Windows- ja Android-toteutusten yksityiskohtia. Lopuksi pohditaan työnteossa ilmentyneitä ongelmakohtia ja esitetään jatkokehitysideoita.

2 NYKYTILANNE

Teleste Oyj:n CATVisor-ohjelmistotuoteperhe on HFC-verkkojen etäiseen ylläpitoon, tarkkailuun ja hallintoon suunniteltu ohjelmistotuoteperhe, joka toimii joko verkon yli tai paikallisella sarjaliikenneyhteydellä. CATVisor ohjelmistot tukevat vain Windows-alustaa. (Teleste 2015a.)

2.1 CATVisor Commander -ohjelmisto

CATVisor Commander on CATVisor-ohjelmistotuoteperheen yksi ohjelmisto, jolla käyttäjä voi: (Teleste 2015a)

- näyttää laitteiden tämän hetkisen tilanteen,
- näyttää ja määrittää laitteiden toimintaparametreja,
- pitää kirjaa moduulien tiloista ja tapahtumista,
- tallentaa ja ladata laiteasetukset XML-muodossa
- ja tulostaa laiteinventaario sekä hätyytystiedot.

2.2 Viewer

Viewerillä, eli katseluohjelmalla tarkoitetaan CATVisor-ohjelmiston kontekstissa yhden tai useamman lähes samanlaiset ominaisuudet ja parametriavaruuden omaavan laitteen laitehallintakirjastoa. Jokaisen uuden laitemallin viewerin eteen joutuu tekemään paljon töitä, ohjelmointia ja testaamista. Dynaaminen viewer todennäköisesti vähentäisi työtakkaa ja pienentäisi testausalueita. Dynaamista vieweriä käyttäen uuden laitetuen tekeminen olisi vain uusien viestimäärittelyjen ja käyttöliittymämäärittelyjen kirjoittamista. Toiminnallisuus riippuisi aina vain muutamista ohjelmistokokonaisuuksista, eri alustojen dynaamisista viewereistä.

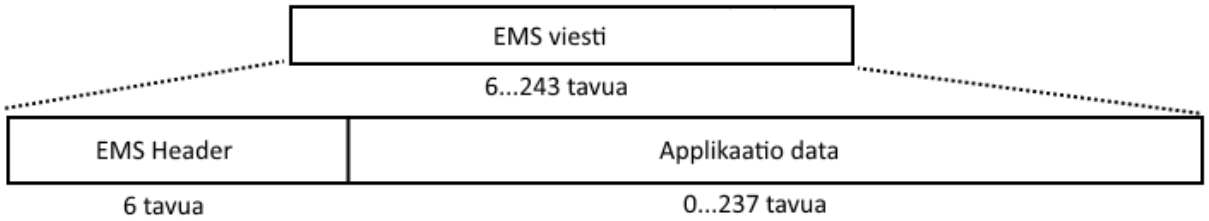
2.3 Android-ohjelmisto

Teleste julkaisi vuoden 2014 alussa laitehallintasovelluksesta Android-version. Sovelluksella pystyy säätämään lähellä olevia laitteita Bluetoothin kautta. Android-versio on suunniteltu lähinnä asennustilanteita varten, joten ohjelma näyttää vain asennustilanteen kannalta tärkeät säätimet. Android-versio tukee tällä hetkellä vain muutamia laitemalleja, kuten ACE8:a. Android-versio on ilmainen. (Teleste 2015b.)

3 VIESTINTÄPROTOKOLLA

EMS-protokolla on kaikkien Teleste laitteiden viestintäprotokollien pohja. Muut laitespesifiset viestit kulkevat EMS-elementtiviestinä. EMS-viesti koostuu headerista ja applikaatiodatasta. (Teleste 2014.)

Kuvassa 1 EMS header määrittelee esimerkiksi lähettäjän käyttämän EMS -protokollaversion, viestityypin, applikaatiotunnuksen ja joitain lippuarvoja. (Teleste 2014.)



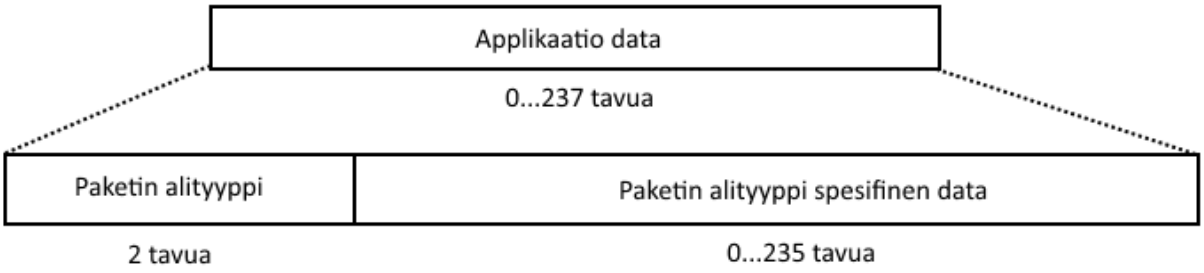
Kuva 1. EMS -viestin rakenne

Viestityyppeihin kuuluvat esimerkiksi: (Teleste 2014.)

- Yleisviesti: viesti, jota kaikki laitteet tukevat, esimerkiksi laitteiden perustietojen kyselyt
- Statusviesti: kertoo laitteen tilasta, esimerkiksi virheraportit
- Elementtiviesti: laitespesifinen viesti

Applikaatiotunnuksen perusteella voidaan yhdistää vastaus ja kysely toisiinsa. Lippuarvot kertovat esimerkiksi, onko kyseessä kysely- vai vastausviesti. (Teleste 2014.)

Kuvassa 2 applikaatiodatan paketin alityypin sisältö vaihtelee EMS headerissä määritellyn viestityypin mukaan, jonka mukaan edelleen paketin alityypin spesifinen data vaihtelee. Yleisviestissä alityyppi on yleiskomento, statusviestissä statuskoodi ja elementissä elementtityyppi.



Kuva 2. Applikaatio datan rakenne

4 TEKNIIKAT

4.1 JSON-tietorakenne

JSON, eli JavaScript Object Notation, on avoimen standardin tietorakenne tiedonvälitykseen. JSON esiteltiin ensimmäistä kertaa vuonna 2001, ja se pohjautuu ECMA-262 standardin 3. painoksessa määriteltyyn JavaScriptin komentosarjakielen osajoukkoon. (EMCA International 2013.)

JSON on kevyt ja yksinkertainen: sitä on koneiden helppo jäsentää ja generoida sitä sekä ihmisten helppo lukea sitä ja kirjoittaa. Tietorakenteen käsitteet ja merkintätavat ovat helposti omaksuttavia C-pohjaisten ohjelmointikielten käyttäjille. (JSON 2015.)

JSON-merkinnät

Kuva 3 kattaa kaikki JSON:n tukemat esitysmuodot. Ensimmäiset neljä arvoa esittävät lukuja JSON:n sallimissa eri muodoissa. Sallittuja esitysmuotoja luvuille ovat kokonaisluku, desimaaliluku ja tieteellinen luku. Lukuarvojen jälkeen on esimerkki merkkijonosta. JSON-merkkijonot voivat sisältää mitä tahansa Unicode-merkkejä ja kenoviivalla aloittavia ohjausmerkkejä. Merkit on voitu kirjoittaa myös nelimerkkisenä heksadesimaalina, jota tulee edeltää ”\u”-ohjausmerkit. Merkkijonot tulee aina asettaa lainausmerkkien sisään. Seuraavat kaksi arvoa ovat totuusarvoja, voivat siis olla vain true tai false. Arvot voivat myös olla tyhjiä eli null.

JSON:ssa käytetään kahta säiliötyyppiä, objektia ja listaa. Näille on määritelty omat aloitus- ja lopetusmerkit. Objektissa ne ovat aaltosulkeet ja listassa hakasulkeet. Säiliöiden sisäiset arvot erotellaan aina pilkulla. Vain objektissa kuuluu jokaisella arvolla olla myös oma tunniste. Tunniste asetetaan merkkijonon tavoin lainausmerkkien sisään. JSON-tietorakenteessa on aina vähintään yksi objekti, kaikista ylimmät määritteet tulevat objektin sisään.

```
{
    "integer": 100,
    "double": -100.0,
    "scientific": -1e100,
    "scientific2": 1E-100,

    "string": "abcd°µ \\" \\ \/ \b \f \n \r \t \u0842"

    "boolean1": true,
    "boolean2": false,

    "nothing here": null,

    "listOfAnything": [
        "string",
        100,
        -1E-100,
        false,
        null,
        { "name": "object", "value": -2.5 },
        [ "object", -2.5 ]
    ]
}
```

Kuva 3. JSON-muodossa olevaa satunnaista dataa

JSON-tietotyyppejä siis ovat luku, merkkijono, totuusarvo, objekti, lista ja null.

Luku ei tarvitse aloitus- tai lopetusmerkkejä, se voi olla kokonaisluku, desimaaliluku tai tieteellinen luku. Merkkijono on asetettava aina lainausmerkkien sisään, se voi koostua Unicode-merkeistä, ohjausmerkeistä tai nelimerkkisistä heksadesimaali-ilmaisuista. Totuusarvo ei tarvitse aloitus- tai lopetusmerkkejä, se voi olla vain joko true tai false. Objekti ja lista voi sisältää mitä tahansa JSON-tietotyyppejä ja arvot on eroteltava toisistaan pilkulla. Lisäksi objekti sisäisiä arvoja tulee aina edeltää lainausmerkeissä oleva tunniste.

4.2 XML-merkintäkieli

XML, eli Extensible Markup Language, on vuonna 1996 julkaistu tiedostomuoto tiedonvälitykseen. XML:stä on useampi kuin yksi versio. Versioiden väliset erot

liittyvät lähinnä merkkien käsittelyyn. Tämän hetkisiä käytettyjä virallisia versioita ovat 1.0 ja 1.1. (W3C 2015a, Introduction.)

Kuvassa 4 XML kostuu elementeistä ja attribuuteista. Elementillä voi olla alielementtejä, src- ja IMG-elementit ovat HYPERLINK-elementin alielementtejä. Attribuutit määrittelevät elementtien ominaisuuksia, esimerkiksi hyperlinkin nimen ja fontin. Elementti vaatii aina aloitus- ja lopetussulkeet. Jos elementillä on alielementtejä tai/ja sisäinen arvo, täytyy elementin lopetussulkeissakin mainita elementtitunniste, kuten HYPERLINK ja src elementeillä. Attribuutit tulevat elementin sulkeiden sisään ja attribuuttiarvot on aina asetettu lainausmerkkien sisään. (W3C 2015a, Logical Structures.)

```
<HYPERLINK name="XML resources" font="bold|italic">
    <src>http://www.w3.org/XML/</src>
    <IMG src="/xml_icon" format="png"/>
</HYPERLINK>
```

Kuva 4. Kuvitteellinen esimerkki hyperlinkin kuvauksesta XML-muodossa

XML-muotoilussa ei ole suoranaista tukea erottaa tietotyyppejä toisistaan, vaan tuki tulee skeemoista.

4.3 Skeema

Skeema on tietorakennemalli, joka määrittelee miten tietyn tyyppisten objektien kuvaus pitää toteuttaa, mitä ominaisuuksia objektille kuuluu ja mitkä ominaisuudet on vähintään määriteltävä. Skeemojen tarkoituksena on helpottaa tiedon jäsentämistä ja varmistaa tiedon oikeellisuus.

4.4 Jäsentimistä

Tietorakennekontekstissa käytetään käsitteitä DOM- (Document Object Model) ja SAX-tyylinen (Simple API for XML) jäsennin, käsitteet tulevat alun perin XML:lle kehitetyistä jäsennysteknologioista.

DOM-jäsennin käsittelee ensin koko tiedoston ja tekee siitä puumallin muistiin. Tämän jälkeen puumallista voidaan hakea ja käsitellä tietoa halutulla tavalla. (Harold 2015, The Document Object Model.)

SAX-jäsennin pohjautuu tapahtumiin. Tapahtuma voi olla esimerkiksi objektin alku- tai loppukohta tietorakenteessa. SAX-jäsennin pitää muistissa vain tapahtumat. Tapahtumille voidaan rekisteröidä erikoistetut käsittelymetodit, ja näin tietorakenne voidaan saada esimerkiksi heti yhden käsittelykerran jälkeen halutussa lopullisessa muodossa. (ORACLE 2015.)

Taulukossa 1 on tiivistetty DOM ja SAX jäsentimien hyvät ja huonot puolet.

Taulukko 1. DOM ja SAX jäsentimien hyvät ja huonot puolet (Harold 2015, Choosing between SAX and DOM.)

DOM-jäsennin
Puumalli on kertakäsittelyn jälkeen aina saatavilla DOM-jäsenin on helpompi toteuttaa kuin SAX
Puumalli voi viedä paljon muistia, eikä välttämättä edes mahdu muistiin
SAX-jäsennin
Suorituskyvyn, nopeuden, kannalta monesti tehokkaampi kuin DOM Käsittely ei käytä paljon muistia
Tietorakenteesta on tehtävä tietorakennetta mallintava luokka

4.5 JSON vai XML

Kuvasta 5 näkee kuinka JSON:ssa käytetään erilaisia merkintätapoja erottamaan useimmissa ohjelmointikielissä käytetyt perustietotyypit toisistaan.

Koneen on helpompi käsitellä JSON:a. XML:n tapauksessa kone joutuu päättelemään tietotyypin tilanteesta riippuen alielementtien tunnisteista tai/ja elementtien attribuuteista. Esimerkiksi kuvassa 5 mallinnetun henkilön vanhat osoitteet ovat tässä tapauksessa esitetty listana. XML:n tapauksessa tiedostoa jäsentävät ohjelmisto ei tiedä tätä vasta kun se on käynyt koko listaelementin läpi ja tarkistanut, että jokaisella alielementillä on sama tunniste. JSON taas on lista vaikka lista-arvoja olisi vain yksi. XML:n tapauksessa myöskään osiin eritellyn osoitteen tapauksessa jäsennin ei voi päätellä nykyisestä tietorakenteesta halutaanko esimerkiksi talonnumero esittää lukuna vai tekstinä. JSON taas tietää, että kyseessä on luku jo heti ensimmäisen numeron käsittelyn jälkeen.

```
{
  "Person": {
    "name": "John",
    "age": "25",
    "sex": "Female",
    "currentAddress": [
      "Osoite 123, ",
      { "textStyle": "bold","text": "Helsinki" },
      ", Finland"
    ],
    "oldAddresses": {
      "address": [
        {
          "state": "USA",
          "city": "New York",
          "street": "Somewhere",
          "number": 123
        },
        "Somewhere 123, New York, USA"
      ]
    }
  }
}
```

```
<Person>
    <Name>John</Name>
    <Age>25</Age>
    <Sex>Female</Sex>
    <CurrentAddress>
        Osoite 123,
        <b>Helsinki</b>
        , Finland
    </CurrentAddress>
    <OldAddresses>
        <Address>
            <State>USA</State>
            <City>New York</City>
            <Street>Somewhere</Street>
            <Number>123</Number>
        </Address>
        <Address>Somewhere 123, New York, USA</Address>
    </OldAddresses>
</Person>
```

Kuva 5. Esimerkki samojen henkilötietojen kuvauksesta JSON:lla (vasen) ja XML:illa (oikea)

JSON:ssa ei voi kuitenkaan joustavasti ilmaista esimerkiksi, että osa tekstistä käyttää erikoismuotoilua. Kuvan 5 XML-esimerkissä nykyisen osoitteen kaupunkimääritteessä osa tekstistä asetettu elementin sisään, jonka tunniste on ”b”-kirjain. Esimerkiksi XML:lla mallinnetuilla nettisivuilla käytetään usein tällaista ilmaisua, tässä tapauksessa tarkoitetaan lihavointia.

XML:n tietokuvauksessa ei ole standardoitua tapaa päätellä perustietotyyppejä. Tietotyyppien määrittelyä varten on olemassa yleisesti käytettyjä XML -skeemoja, mutta niiden käyttö on valinnaista.

XML:n hyvänä puolena on joustava muotoilu ja huonona puolena toistuvat tunnisteet. JSON:n etuina ovat selkeästi määritellyt perustietotyypit ja niiden merkintätavat. Ne tekevät JSON:sta helppo lukuisen sekä ihmiselle että tietokoneelle.

4.6 COM-teknologia

COM, eli Component Object Model, on yksi Microsoftin kehittämistä perustusteknologioista, jonka avulla ohjelmakomponentit voivat kommunikoida keskenään. Nämä ohjelmakomponentit ovat tunnetumpia DLL-päätteisinä tiedostoina. Komponentit voivat sijaita samassa tai eri prosesseissa, jopa eri tietokoneilla.

Vastaavia Microsoftin tekniikoita ovat OLE, OLE2 ja ActiveX, mutta nykyään ActiveX-komponentti on vain yleisnimitys näistä Microsoftin teknologioista. Myös arkipäivässä käytössä olevat Office-ohjelmistot hyödyntävät näitä teknologioita käyttäessään toistensa ominaisuuksia, esimerkiksi silloin kun Wordiin lisätään Excel-taulukko tai toisinpäin, kun Exceliin lisätään tekstikenttä. (MSDN Development Center 2015f; MSDN Development Center 2015g.)

COM-komponentit eivät paljasta kaikkia metodejaan vaan niitä käytetään komponenttien implementoimien rajapintojen kautta. (MSDN Development Center 2015b)

Näitä jaettuja komponentteja ei voi eritellä pelkällä nimikonventiolla, sillä silloin voisi aiheutua päällekkäisyyksiä, eikä kone tietäisi, mitä komponenttia varmasti tarkoitettaisiin. Ongelma on ratkaistu generoidulla ainutkertaisella tunnusnumerolla eli GUID:lla (Globally Unique Identiefier), joka täytyy rekisteröidä koneelle ennen käyttövalmiutta. Samalla GUID:lla ei voi rekisteröidä yhtäaikaisesti eri komponentteja tai eri versioita komponentista. (MSDN Development Center 2015d.)

Isompia COM-komponenttikokonaisuuksia suunniteltaessa voidaan käyttää hyväksi koodin uudelleenkäyttömenetelmiä, kuten delegointia ja aggregointia, jotka toimivat samalla periaatteella kuin oliokeskeisissä kielissä periytyminen.

COM-konseptissa näiden termien selittämiseksi käytetään käsitteitä ulkoinen ja sisäinen olio, jotka ovat verrattavissa oliokeskeistenkielien yli- ja aliluokkiin. (MSDN Development Center 2015e.)

Delegoinnin yksinkertainen tapaus on sisältyvyys, jossa ulkoinen olio sisältää sisäisen olion ja delegoi metodeissaan osan työstä sisäiselle oliolle kutsumalla tämän metodia. Samalla tavalla kuin olio-ohjelmoinnissa luokka voi sisältää muita luokkia ja käyttää tämän metodeja. (MSDN Developement Center. 2015c)

Aggregointi on delegoinnin erikoistapaus, jossa ulkoinen olio paljastaa osan rajapinnoistaan sisäiselle oliolle ja antaa tämän implementoida ne. Aggregointi on hyödyllistä, kun useampi ulkoinen olio tulee käyttää samoja implementaatioita rajapinnoista. Tämä on verrattavissa olio-ohjelmoinnin yli- ja alaluokka väliseen periytymiseen. (MSDN Development Center 2015a.)

4.7 ATL ja WTL kirjastot

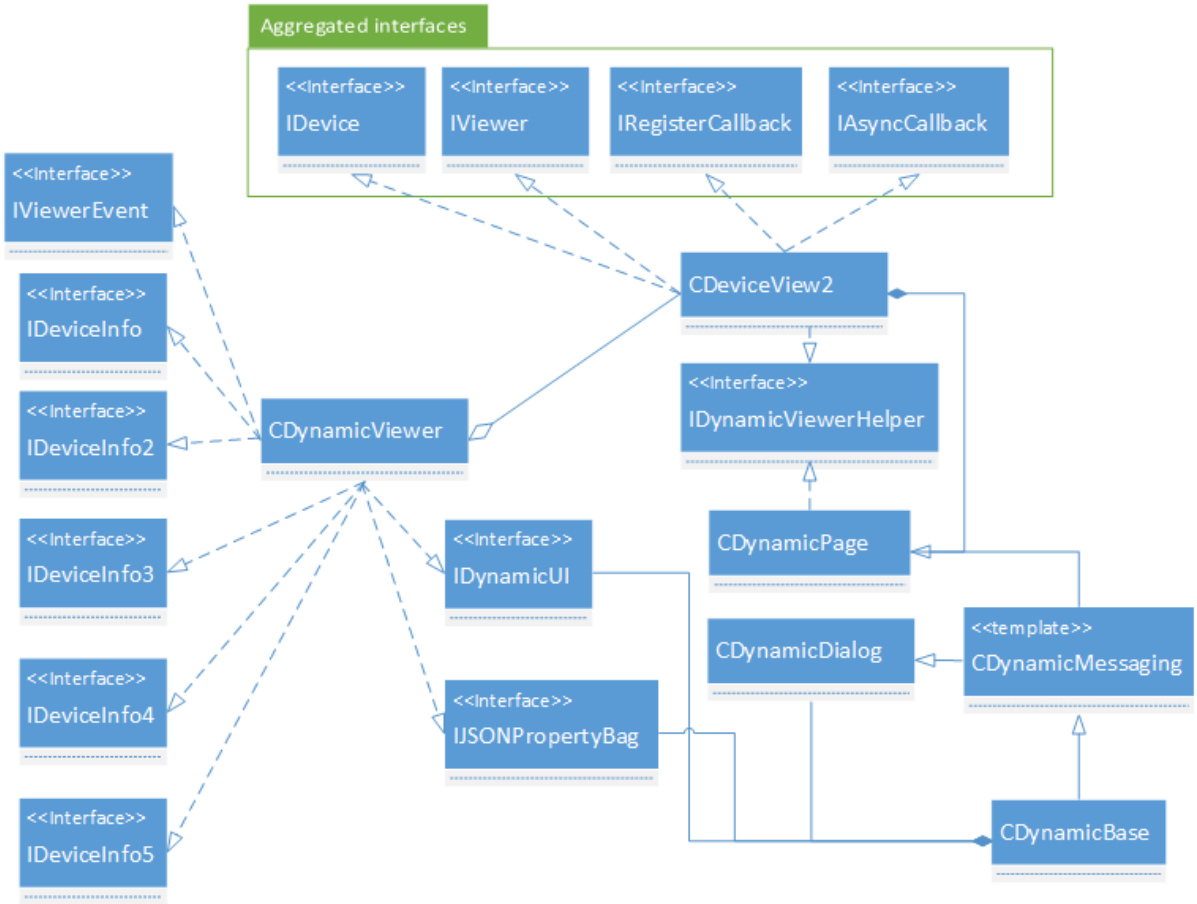
ATL, eli Active Template Library, on Microsoftin kehittämä generoituihin luokkiin perustuva kirjasto, jonka on tarkoitus helpottaa COM-luokkien kehittämistä. Se sisältää valmiita pohjia yleisimmin tarvituille COM-luokille, muistinhallintaa helpottavia luokkia ja hyödyllisiä makrofunktioita. ATL on kevyempi vaihtoehto MFC:lle (Microsoft Foundation Classes), joka oliokeskeinen sovitin Win32 API:lle ja on tarkoitettu enemmänkin käyttöliittymällisten sovellusten nopeaa kehittämistä varten. Nykyään Microsoft ei kumminkaan suosittele MFC-kehittämistä, koska samanlaista kehittämistä voi tehdä .NET kirjastoilla. COM ja ATL ovat kumminkin vielä käytössä niiden keveyden ja tehokkuuden takia. ATL:n ehkä hyödyllisimpiä luokkia ovat viisaat osoittimet, jotka hoitavat COM-rajapintojen osoittimien muistinhallinnan yleisimmissä tilanteissa. (MSDN Developer Network 2015a; MSDN Developer Network 2015b.)

WTL, eli Windows Template Library, on Microsoftin tekemä ATL-kirjaston laajennus, joka oli tarkoitettu aluksi Microsoftin sisäiseen käyttöön, mutta laitettiin myöhemmin jakoon tuettomana lisäosana MS Visual Studioon. (MSDN Magazine 2007.)

5 COMMANDERIN RAKENNE

Dynaaminen laitehallintakirjaston toteuttaminen ei vaadi muutoksia Commanderin rakenteeseen eikä toimintaan. Dynaamisuuteen tarvittava logiikka voidaan toteuttaa samalla lailla omana kirjastonaan kuin mikä tahansa muukin laitehallintakirjasto.

Kuvassa 6 aggregoidut rajapinnat ovat kaikille viewereille yhteisiä toteutuksia, DynamicViewer-luokka antaa DeviceView2-luokan implementoida ne. Nämä rajapinnat pitävät huolta esimerkiksi Commanderin ja viewerin sivujen välisestä kommunikaatiosta. Vasemman puolen DeviceInfo- ja ViewerEvent-rajapintojen toteutukset ovat viewerikohtaisia. Nämä antavat Commanderille erilaisia laitteeseen liittyviä hyödyllisiä tietoja.



Kuva 6. UML-kaavio Commanderin viewerin kannalta olennaisten luokkien ja dynaamisen viewerin luokkien suhteista

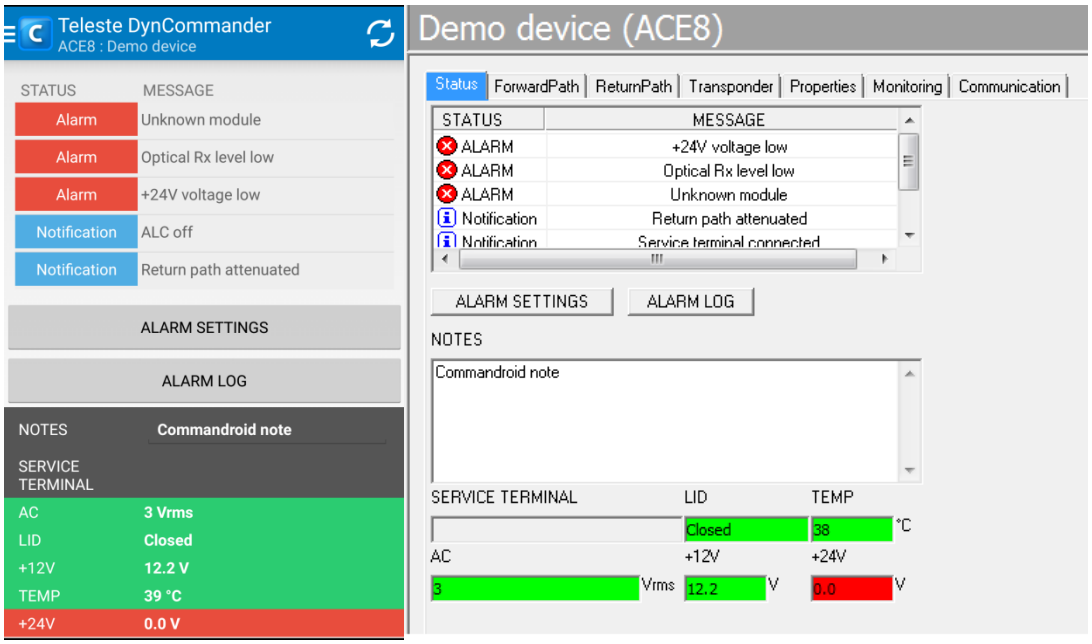
Dynaamisen viewerin tukemiseksi Commander projektin DeviceView2-luokkaan täytyi tehdä yksi muutos: DynamicViewerHelper-rajapinnan lisääminen ja toteuttaminen. Rajapinnan avulla dynaaminen viewer voi esittää dynaamisuuden kannalta tärkeät rajapintansa DeviceView2-luokalle, joka taas voi esittää ne viewerin dynaamisille sivuille.

Dynaamisuuden kannalta tärkeitä uusia rajapintoja ovat DynamicUI ja JSONPropertyBag. JSONPropertyBag-rajapinta mahdollistaa tietorakenteista jäsennettyjen käyttöliittymämääritelmien kyselyn. DynamicUI-rajapinta mahdollistaa uniikin kontrolliresurssitunnisteen kyselyn kontrolleja alustaessa, sekä viestien generoinnista viestimääritelmistä kontrollin pyytäessä. Tämä rajapinta on toteutettu viewerin puolella, jotta jokaisen sivun ei tarvitsisi kopioida viestimääritelmiä, ja ettei sivujen välissä tulisi resurssitunnisteiden päällekkäisyyksiä.

6 TYÖN TOTEUTUS

Tässä luvussa käydään läpi työn toteutuksen eri vaiheita: käyttöliittymämäärittelmien suunnittelua, viestien tietorakennetta ja erikseen Windows ja Android toteutuksien yksityiskohtia.

Kuvassa 7 näkyy valmiin prototyypin generoima näkymä ACE8-laitteen statussivusta. Liitteestä 1 löytyy sivun generoimiseen käytetyt käyttöliittymämääritelmät. Määritelmien massiivisuuden tähden määritelmistä on poistettu Windowsin käyttämät kontrollien koko- ja marginaalimääritelmät.



Kuva 7. Tietorakenteen määritelmistä generoidut viewerien Status sivut ACE8-mallin laitteelle. Vasemmalla Android sovellus ja oikealla Windows sovellus

6.1 Käyttöliittymän tietorakenne

Tietorakenteiden suunnittelua varten täytyy ensin tietää mitä kontrolleja laitehallintakirjastot käyttävät. Telesten laitehallintakirjastoissa käytetään pitkälti peruskontrolleja: muokkausruutu, tekstiruutu, valintaruutu, valintanappi, painike, pudotusvalikko, liukusäädin, taulu ja lista.

Kaikki peruskontrollit eivät pysty toimimaan itsenäisesti, esimerkiksi:

- Valintanapit toimivat yleensä ryhmissä.
- Liukusäätimet muokkaavat erillisen muokkausruudun tai tekstiruudun arvoa.
- Painikkeen painallus avaa mahdollisesti dialogin, toimii merkkinä aloittaa jonkun toiminnan tai muokkaa jotain arvoa määritellyllä tavalla.

Osia kontrollin välisiä toiminnallisia suhteita on vaikea ja osittain mahdoton mallintaa tietorakenteessa. Tämän takia yleisimmistä kontrolliryhmistä kannattaa toteuttaa uusi kontrolli, jota voidaan mallintaa tietorakenteessa. Näin vältytään myös kontrollien suhteiden määrittelyltä, mikä tekee myös tietorakenteesta helpomman kirjoittaa ja lukea yksinkertaistamalla sitä.

Taulukon 2 uloimmista määrityksistä (root) mappingFile, deviceDescFile ja flagSpecFile, ovat viittauksia muihin tietorakenteisiin. MappingFile sisältää laitteen käyttämät parametrimääritykset, deviceDescFile viestimääritykset ja flagSpecFile laitteen käyttämät lippuarvot ja niiden kuvaukset.

Taulukko 2. Laitetta kuvaavan tietorakenteen perusmääritykset

Type	Definitions	Type	Description
root	mappingsFile	string	defines the name of parameter definition json file
	deviceDescFile	string	defines the name of message definition json file
	flagSpecFile	string	defines the name of flag specification xml file
	pages	array of page objects	pages are defined in array because the order of pages matters
	dialogs	object with dialog object definitions	
page /dialog	title	string	title of page, shown in tab selection
	profile	string	defines minimum user profile needed to use the page
	rows	array of rows	defined left to right, order single row is an array of control definitions
control	type	string	rows,groupBox,textBox,coloredTextBox, editBox ,textbox ,comboBox,listCtrl, button, adjustmentSlider, radioGroup, valuedRadioButton
	properties	object with property definitions	list of properties to define control functionality

Sivu, dialogi ja kontrolli määritykset ovat uusia. Määritelmät on suunniteltu Teleste Commander käyttö spesifisiksi, eivätkä siis sovellu yleiseen sovelluskehittämiseen. Tarkemmat kontrollikohtaiset määritteet löytyvät liitteestä 2.

Tietorakenne nimetään laitteen GUID:lla ainutkertaisella numerosarjalla, jota kysellään laitekirjastoa käynnistäessä ja tämän avulla voidaan valita laitteelle oikea tietorakenne tiedosto.

Taulukosta 3 nähdään, että kaikille kontrolleille voidaan määritellä ulkonäöllisiä muuttujia, kuten leveys, korkeus ja marginaali. Android-versiossa ulkonäöllisiä määrityksiä ei kumminkaan huomioida, koska Androidin kontrollit skaalatutuvat automaattisesti sisällön mukaan.

Taulukko 3. Kontrollien yhteiset määritykset, jotka määritellään kontrollin properties-objektin muuttujina

Type	Definitions	Type	Description
common for all ctrls	margin	integer	sets all margins to defined value
	marginLeft	integer	sets left margin to defined value
	marginRight	integer	sets right margin to defined value
	marginTop	integer	sets top margin to defined value
	marginBottom	integer	sets bottom margin to defined value
	title	string	
	titleWidth	integer	pixels,dp,used only if title is defined
	titleHeight	integer	pixels,dp,used only if title is defined
	titlePlacement	string	placement of title, left,right,top,bottom
	postFix	string	postfix after field
	valueMessage	string	name of value message
	statusMessage	string	name of status message
	useStatusMessage	boolean	default: true, if false, uses flags if defined
	getParameter	string	parameter name in response
	flags	array of integers	flag indexes in order of importance

Viestimäärityksiä jokaisella kontrollilla voi olla kaksi: arvo ja tila. Arvoviesti määrittelee viestin, josta saadaan kontrollin arvo, ja tilaviesti kertoo onko esimerkiksi arvo varmasti toiminnallisessa tilassa tai ylittää/alittaako arvo määritellyt virhemarginaalit. Tilaviesti voidaan määritellä joka kontrollille, mutta Windows-

versiossa ainoastaan coloredTextBox-kontrollityyppi voi näyttää tilaviestin taustavärinä.

Kaikkien viitattujen viestien täytyy olla määritelty viestimäärittelyissä, jotka löytyvät Taulukon 2 deviceDescFile-määrityksessä viitatussa tiedostosta.

Tilaviestille vaihtoehtoisesti voidaan käyttää myös lippuarvoja, joita voidaan tilaviestistä poiketen määritellä useampi kuin yksi. Useamman lippuarvon omaavan kontrollin tila määräytyy lippuarvojen korkeimman virhetilan mukaan.

Kuvasta 7 näkyy myös kuinka Android-sovelluksessa ei oteta huomioon kontrollien koko- ja marginaalimääritelmiä, kaikki kontrollit ovat Android-sovelluksessa saman levyisiä, ja skaalatutuvat pituussuunnassa sisällön mukaan. Windows viewerin kannalta kokomääritelmät ovat sen sijaan pakollisia.

6.2 Viestien tietorakenne

Telesten nykyisessä Android-laitehallintasovelluksessa on jo käytössä viestien kuvaus JSON tietorakenteena ja sille on valmiiksi määritelty formaatti. Dynaaminen viewer käyttää samaa formaattia, koska ei ole tarvetta uudelle. Viestimääritelmät on toteutettu kahtena tietorakenteena, toinen kuvaa parametriarvot ja toinen näitä käyttävät viestirungot.

Kuvassa 8 on yksittäinen viestirunko, joka koostuu kyselystä ja sen vastauksesta. Kysely ja vastaus koostuvat: viestityypistä ja tietosisällöstä. Tietosisältö taas määrittelee kaiken tiedon, joka kyselyyn tai vastaukseen tule mukaan.

```
"message_name": {
 "description":"Message description",
 "request": {"messageType":"MESSAGE_BASE_TYPE",
  "payload": [
   {"type":"parameter","name":"subaddress","constant":"MODULE_1"},
   {"type":"parameter","name":"parameter","constant":"MODULE_VALUE1"},
   {"type":"parameter","name":"valueCount","value":"0x02"}
  ]
 },
 "response":{"messageType":"ELEMENT",
  "payload": [
   {"type":"uint8","name":"STATUS"},{"type":"uint8","name":"COUNT"},
   {"type":"uint16","name":"value1","format":{"type":"numeric"}},
   {"type":"uint16","name":"value2","format":{"type":"numeric"}}
  ]
 }
}
```

Kuva 8. Esimerkki viestirungon JSON muotoisesta määritelmästä

Taulukossa 4 on kuvattu kuvan 8 määritelmien mukaan generoidun viestin rakennetta. Ylempi rivi on kyselyviesti. Alempi rivi on kyselyn todennäköinen vastausviesti kyselyn onnistuessa virheittä ja laiteen tuettaessa kyseltyä parametria.

Taulukko 4. Esimerkki viestimääritelmästä generoidusta viestistä

	2 bytes	1 byte	1 byte	1 byte	1 byte
EMS Header	ELEMENT TYPE	GET PARAM	SUBADDRESS	START	COUNT

	2 bytes	1 byte	1 byte	2 bytes	2 bytes
EMS Header	ELEMENT TYPE	STATUS	COUNT	VALUE 1	VALUE 2

6.3 JSON-jäsennin

Telesten nykyisessä Android-sovelluksessa käytetään SAX-pohjasta Jackson nimistä JSON-jäsennintä. Jackson sallii myös tarpeessa kommentoinnin tietorakenteen syntaksien välissä ja kaikkien tietotyyppien kirjoittamisen tekstimuodossa. Nykyisessä Android-sovelluksessa on käytetyttä hyväksi molempia ominaisuuksia. Tämä on otettava huomioon Windows-puolen jäsentimen valitsemisessa, jos käytetyt jäsentimet eivät tue samoja ominaisuuksia jouduttaisiin tietorakenteistakin tekemään erilliset versiot molemmille alustoille.

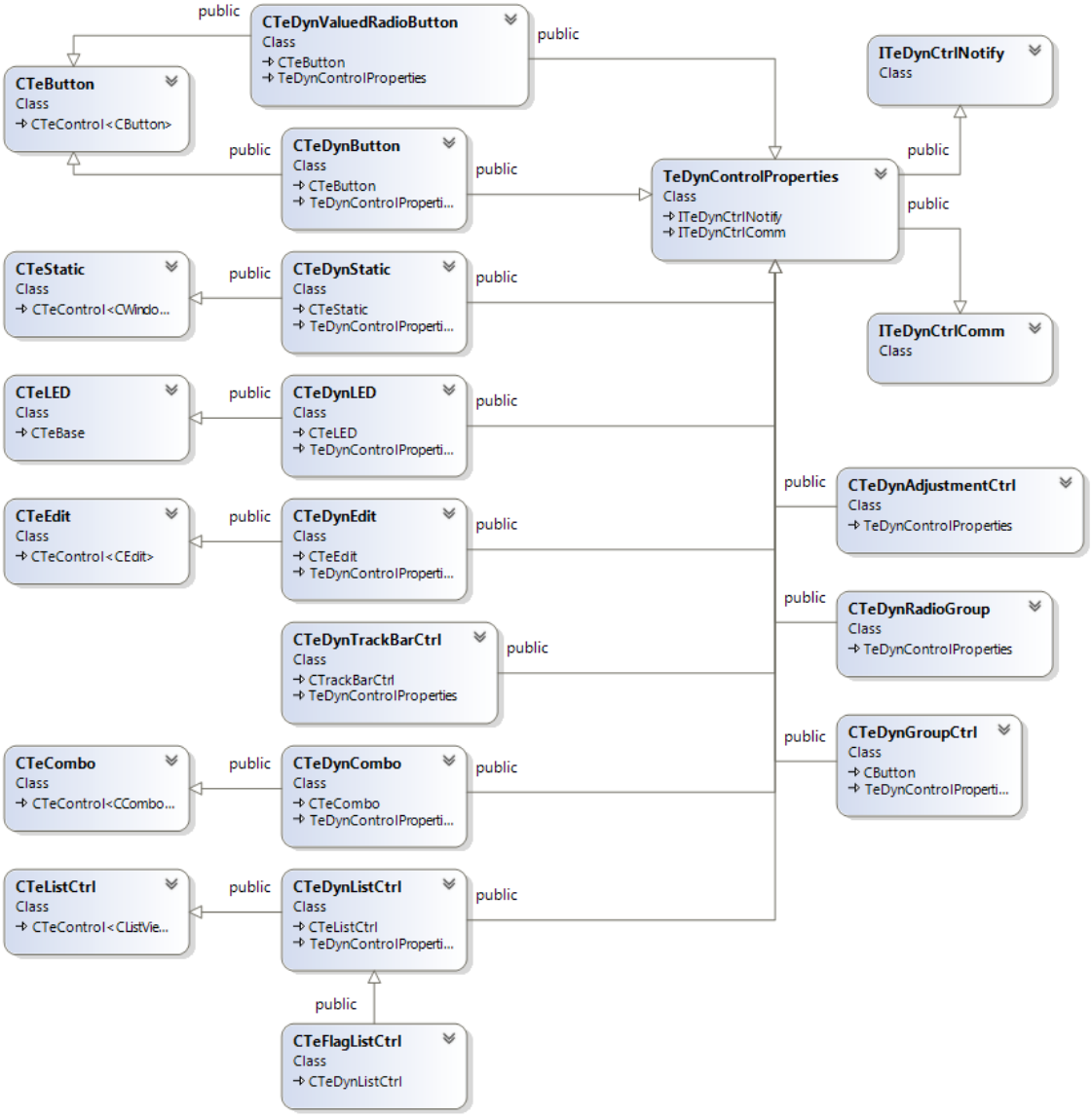
C++:lle on tarjolla vain kolmannen osapuolen jäsentimiä, Microsoft ei tarjoa JSON:n käsittelyyn C++:lla yhtään ratkaisua. Kolmannen osapuolen jäsentimistä rapidjson niminen jäsennin vaikuttaa suorituskyvyn kannalta pätevimmältä. Se myös tukee molempia SAX- ja DOM-tyylistä jäsentämistä. Useissa testeissä rapidjson on myös osoittautunut nopeimmaksi jäsentimeksi, mikä C++:lle on saatavilla. Dokumentaatiossa ei ole kumminkaan mitään mainintaan tunnettujen tietotyyppien käsittelystä tekstimuodossa. (Rapidjson 2015.)

Dokumentaation puutteen vuoksi oli varmempaa kirjoittaa uusi JSON:n jäsennin. Kyseessä on ohjelmiston prototyyppi, joten on viisaampaa kirjoittaa pienemmän työmäärän vaativa DOM-jäsennin. Tietorakenteet eivät myöskään ole niin suuria, etteivätkö ne mahtuisi nykystandardin tietokoneen muistiin.

Liitteessä 3 on kuvattu toteutetun jäsentimen toimintaa. Jäsennin etsii ensin uloimman JSON-objektin alun, minkä jälkeen varsinainen jäsennys alkaa. Jäsennys etenee kirjain kerrallaan alusta loppuun, tarkastaen aina tällä hetkellä käsittelyä sallittavat merkit ja odottaen tietotyyppien alku- ja loppumerkkejä. Alkumerkin saadessa jäsennin kutsuu tietotyypin käsittelyfunktiota. Käsittelyfunktio lukee tietoa, kunnes se löytää loppumerkin.

6.4 Dynaamisen käyttöliittymän toteutus Windowsilla

Telesten viewereissä käytetään TeControls-kirjastossa määriteltyjä kontrolleja, jotka pohjautuvat peruskontrolleihin. Näille kontrolleille tehtiin dynaamisuutta varten näitä laajentava TeDynControls-kirjasto. Näissä laajennuksissa määritellään viestinnän ja dynaamisuuden kannalta olennaisia ominaisuuksia ja rajapintoja. Kuvassa 9 TeControls-kontrollit ovat vasemmalla ja TeDynControls-kontrollit ja niiden rajapinnat oikealla.



Kuva 9. UML-kaavio TeControls- ja TeDynControls-kirjastoista ja niiden rajapinnoista

6.4.1 Kontrollien generointi

Dynaamisen viewerin käynnistyessä ensimmäisenä kysellään laitteen GUID. Tästä numerosarjasta päätellään laitteelle sopiva JSON-tiedosto. Tiedosto ja sen viittaamat muut tietorakenteet jäsennetään. Viestimääritelmät käännetään yleisistä JSON-olioista TSEMP-kirjaston olioiksi, minkä jälkeen viestimääritelmien puumallit voidaan poistaa muistista. Käyttöliittymämääritelmistä sivukohtaiset puumallit jaetaan sivuille sivukohtaisesti ja yleiset dialogimääritelmät jokaiselle

sivulle, koska mikä tahansa sivu voi kutsua näitä dialogimääritelmiä. Sivun aktivoituessa sivu käsittelee saamansa käyttöliittymämääritelmät puumallista ja luo niiden mukaiset kontrollit sekä esittää ne sivun käyttöliittymässä.

6.4.2 Viestiketju

Koska viestit täytyy generoida laitteen viestimääritelmistä, tarvittiin näitä määritelmiä käsittelevä kirjasto. Telesten nykyisen Android-sovelluksen TSEMP-kirjasto pystyy jo tähän, joten ei ole järkeä tehdä kokonaan uutta. Java-ohjelmointikielellä kirjoitettu TSEMP-kirjaston viestimääritelmiä käsittelevä osa käännettiin C++-ohjelmointikielelle.

ATL/WTL ympäristössä käyttöliittymässä tapahtuvat muutokset, kuten tekstinkentän muuttuminen, tulevat sivulle tapahtumina ja huomautuksina. Tapahtumat ja huomautukset voidaan ohjata sivulla määriteltyihin funktioihin. Funktioiden täytyy noudattaa ennalta määriteltyjä muuttujia, kummastakin huomautuksesta ja tapahtumasta saadaan kumminkin ulos ainakin tapahtumaa koskevan kontrollin tunniste. On siis olennaista, että kontrolleja tehtäessä kontrollien tunnisteista ja rajapinnoista tehdään kartta. Kartan avulla huomautuksen tai tapahtuman lauetessa käsittely voidaan ohjata oikealle kontrollille.

TeDynControls-kirjaston kontrollit hoitavat viestin kontrollikohtaisesti, toisin kuin normaalissa viewerissä, jossa viestit käsitellään sivukohtaisesti. Sivun päivittäessä sivu kutsuu sen kontrollien ITeDynCtrlComm-viestintärajapinnan viestilähetysfunktiota, jotka ovat kontrollikohtaisia. Kontrollin lähettäessä viestin sivu lisää viestitunnuksen ja kontrollin viestintärajapinnan osoittimen sivun ylläpitämään karttaan, jonka avulla sivu osaa ohjata kontrollin lähettämän viestin vastauksen oikealle kontrollille käsiteltäväksi.

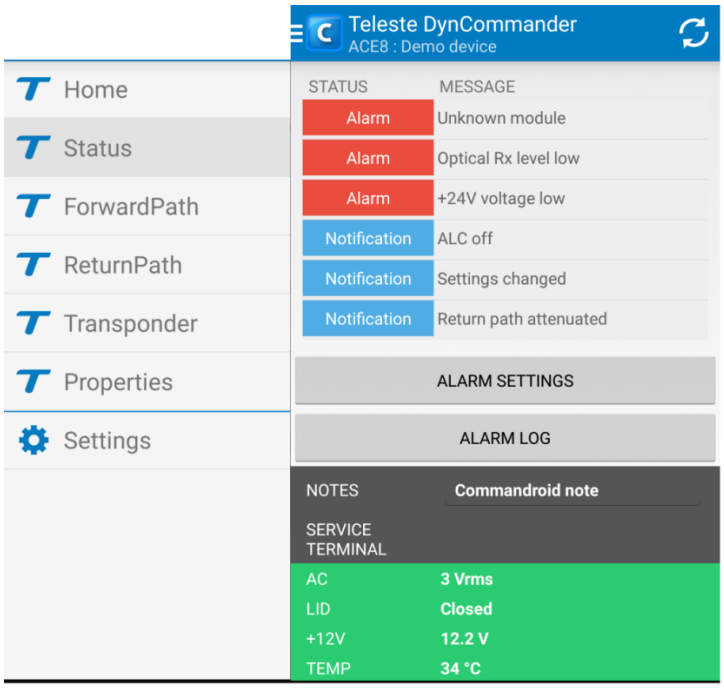
6.5 Dynaamisen käyttöliittymän toteutus Androidilla

Android-versio dynaamisesta vieweristä oli paljon helpompi toteuttaa kuin Windows-versio, koska nykyinen Android-versio sovelluksesta käyttää juuri sellaista

kommunikointityyliä, jota dynaamisuutta varten tarvitaan, eli kontrollikohtaista kommunikointia. Android-sovelluksen kommunikaatioon ei siis tarvinnut kiinnittää huomiota sovellusta tehtäessä.

Android-sovellusta varten tehtiin käyttöliittymämääritelmiä mallintavat luokat ja näiden jäsentäminen tietorakenteesta lisättiin osaksi nykyisien laitemäärityksien jäsennystä.

Liitteen 4 UML-kaavio kuvaa dynaamisen Android-sovelluksen toimintaa. Jäsennin kääntää sivu- ja dialogimääritelmät suoraan DynamicSectionFragment-komponenteiksi, joista ohjelma saa tietää sivujen otsikot ja sivukohtaiset käyttöliittymämääritelmät, joiden perusteella sivu kuuluisi rakentaa. Tämän jälkeen sovellus lisää nämä komponentit sivuvalikkoon. Käyttäjän avatessa sivun se käsitellee sivukohtaiset käyttöliittymämääritelmät ja näyttää niiden mukaiset kontrollit. Kuvassa 10 on esimerkki tässä prosessissa generoidusta sivuvalikosta ja yhden sivun käyttöliittymästä.



Kuva 10. Generoitu ACE8-laitteen Status-sivu ja sivuvalikko

7 YHTEENVETO

Opinnäytetyön tavoitteena oli mallintaa Teleste viewereiden kontrollien yleisimmät ulkonäölliset ominaisuudet ja viestintämääritykset tietorakenteena, joita voitaisiin hyödyntää sekä Windowsilla että Androidilla.

Työn tekoa helpotti kokemus työskentelystä Androidilla, sillä Androidin käyttöliittymät yleensä mallinnetaan XML-tietorakenteena. Alussa oli jo käsitys siitä, miltä komponentti kuuluisi näyttää tietorakenteessa mallinnettuna.

Käyttöliittymän määritelmiä työstäessä pitää ottaa huomioon useita eri käyttötilanteita. Määritelmät pitää olla jatkokäyttäjien kannalta yksinkertaisia ja helppoja kirjoittaa, mutta samaan aikaan kaikki määritelmät eivät voi olla liian yksinkertaisia. Esimerkiksi, jos kontrollin toiminta riippuu useammasta parametreista, pitää ottaa huomioon, että laitteiden viestiavaruudet voivat vaihdella. Kaikkia haluttuja parametreja ei voi ehkä kysellä yhdellä viestillä, jolloin viestit pitää pystyä määrittelemään taulukkona.

Ratkaisu kontrollien määrittelemisestä riveittäin helpottaa järjestelyä. Määritelmiä kirjoitettaessa ei tarvitse kiinnittää huomiota muiden kontrollien sijaintiin, koska ohjelmisto esittää ne määritetyssä järjestyksessä ylhäältä alaspäin ja vasemmalta oikealle. Rivimäärittelyssä on kuitenkin huonona puolena se, että tietorakenteesta voi tulla syvä, jos sisäkkäisiä rivityskontrolleja joudutaan käyttämään. Tällöin myös tietorakenteen luettavuus kärsii.

Opinnäytetyön tuloksena oli toiminnallisuuden kannalta toimiva prototyyppi dynaamisesta vieweristä. Sekä Android- että Windows-versio pystyvät käsittelemään EMS 5 -viestintäprotokollan tyylisiä viestintämääritelmiä ja pystyvät hyödyntämään yhteisiä tietorakenteita.

Mikäli sovellukselle halutaan tehdä jatkokehitystä, on tarpeen selvittää, tukevatko nykyiset määritelmät vanhojen laitteiden viestiavaruuksia. Prototyyppi pohjautuu pitkälti ACE-laitteiden tarpeisiin.

LÄHTEET

EMCA International 2013. Introduction. EMCA-404, The JSON Data Interchange Format, 1. painos. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

ECMA International 2011. EMCA-262, ECMASript Language Specification, 5.1. painos. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

Harold, E. R. 2003. Processing XML with Java. http://www.cafeconleche.org/books/xmljava/

JSON 2015. Introducing JSON. Viitattu 14.4.2015 http://json.org/

MSDN Development Center 2015a. Aggregation. Viitattu 14.4.2015 https://msdn.microsoft.com/en-us/library/windows/desktop/ms686558(v=vs.85).aspx

MSDN Development Center 2015b. COM Objects and interfaces. Viitattu 14.4.2015 https://msdn.microsoft.com/en-us/library/windows/desktop/ms690343(v=vs.85).aspx

MSDN Developement Center 2015c. Containment/Delegation. Viitattu 14.4.2015. https://msdn.microsoft.com/en-us/library/windows/desktop/ms683706(v=vs.85).aspx

MSDN Development Center 2015d. Interface Pointers and Interfaces. Viitattu 14.4.2015. https://msdn.microsoft.com/en-us/library/windows/desktop/ms688484(v=vs.85).aspx

MSDN Development Center 2015e. Reusing Objects. Viitattu 14.4.2015 https://msdn.microsoft.com/en-us/library/windows/desktop/ms678443(v=vs.85).aspx

MSDN Development Center 2015f. The Component Object Model. Viitattu 14.4.2015. https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363(v=vs.85).aspx

MSDN Development Center 2015g. Component Object Model. Viitattu 14.4.2015. https://msdn.microsoft.com/en-us/library/windows/desktop/ms680573(v=vs.85).aspx

MSDN Developer Network 2015a. Introduction to ATL. Viitattu 14.4.2015 https://msdn.microsoft.com/en-us/library/hdf7fy18.aspx

MSDN Developer Network 2015b. Recommendations for Choosing Between ATL and MFC. Viitattu 14.4.2015 https://msdn.microsoft.com/en-us/library/bk8ytxz5.aspx

MSDN Magazine 2007. Windows Template Library 8.0. https://msdn.microsoft.com/en-us/magazine/cc163305.aspx

ORACLE 2015. Java Documentation: Introduction to JAXP. Viitattu 24.4.2015. https://docs.oracle.com/javase/tutorial/jaxp/intro/index.html

Rapidjson 2015. Viitattu: 15.4.2015 https://github.com/miloyip/rapidjson

Teleste 2014. EMS Protocol Description. Versio 1.36. 28.10.2014

Teleste 2015a. CATVisor Commander. Viitattu 24.4.2015 http://www.teleste.com/products/broadband-network/catvisor-management-software/catvisor-commander

Teleste 2015b. Teleste Commander for Android. Viitattu 24.4.2015. http://www.teleste.com/news/2014/teleste-commander-android-now-available-google-play

W3C 2015a. Extensible Markup Language, W3C Working Draft. Viitattu 24.4.2015 http://www.w3.org/TR/WD-xml-961114.html

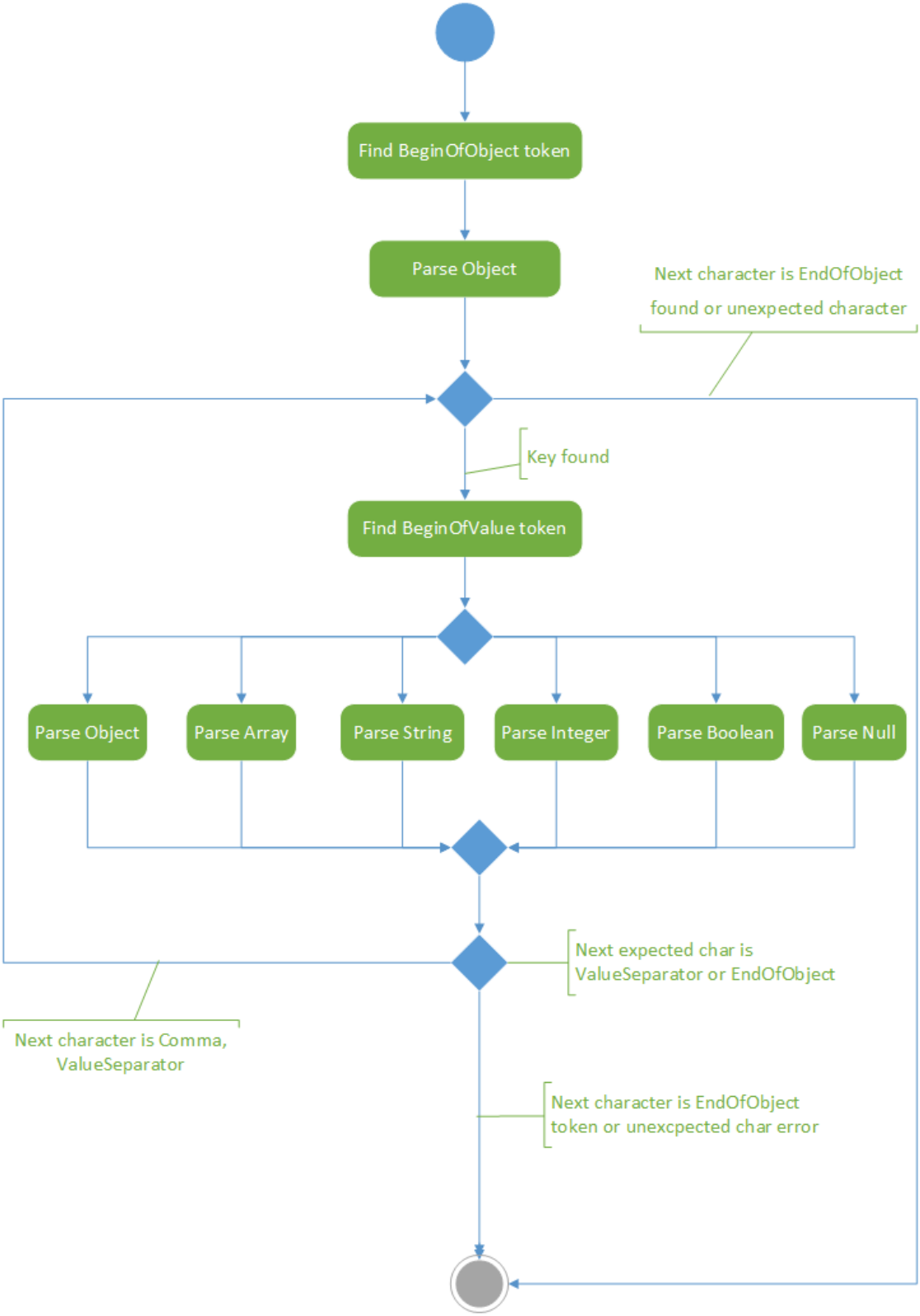
ACE8 Status-sivun käyttöliittymämääritelmät


```
{"type":"flagList",
 "properties":{
    "labels":[
        {"title":"STATUS"},{"title":"MESSAGE"}
    ]
 }
},
{"type":"rows",
    "rows":[
        [{"type":"button","properties":{"title":"ALARM SETTINGS","openDialog":""}}],
        [{"type":"button","properties":{"title":"ALARM LOG","openDialog":""}}]
    ]
},
{"type":"editBox","properties":{"title":"NOTES","multiline":true,"verticalScroll":true,"valueMessage":"EMS_GET_NOTES"}},
{"type":"rows",
 "rows":[
    [
        {"type":"coloredTextBox","properties":{"title":"SERVICE TERMINAL","statusMessage":"","valueMessage":""}},
        {"type":"coloredTextBox","properties":{"title":"AC","statusMessage":"station_rms_voltage_status","valueMessage":"station_rms_voltage","postFix":"Vrms"}}
    ],[
        {"type":"coloredTextBox","properties":{"title":"LID","statusMessage":"lid_status","valueMessage":"lid_state","useStatusMessage":false,"flags":[18]}},
        {"type":"coloredTextBox","properties":{"title":"+12V","statusMessage":"station_12v_voltage_status","valueMessage":"station_12v_voltage","postFix":"V"}}
    ],[
        {"type":"coloredTextBox","properties":{"title":"TEMP","statusMessage":"station_temperature_status","valueMessage":"station_temperature","postFix":"°C"}},
        {"type":"coloredTextBox","properties":{"title":"+24V","statusMessage":"station_24v_voltage_status","valueMessage":"station_24v_voltage","postFix":"V"}}
    ]
  ]
}
```

Taulukko komponenttikohtaisista määritelmistä

editBox	readOnly	boolean	defines if value is read only, can only be observed
	multiline	boolean	defines if value can go on multiple lines in case there's no horizontal space
	verticalScroll	boolean	defines whether vertical scrollbar is shown
	horizontalScroll	boolean	defines whether horizontal scrollbar is shown
	numberOnly	boolean	defines whether value can consist only of numbers
coloredTextBox	multiline	boolean	defines if value can go on multiple lines in case there's no horizontal space
textBox	multiline	boolean	defines if value can go on multiple lines in case there's no horizontal space
comboBox	list	array of objects { "title", "value" }	defines the list shown and the actual data used to save selection
listCtrl	labels	array of label objects { "title", "width" }	titles for each column and the initial width of column
	rows	array of row objects {index,title,valueMessage,fields}	defines fixed index position of row, title (first column), message to used to get the row data, and the payload names for each column data
groupBox	controls	array of control objects	
button	multiline	boolean	defines if title can go on multiple lines in case there's no horizontal space
	openDialog	string	name of dialog to open
checkBox	offValue	integer	defines value saved when not checked
	onValue	integer	defines value saved when checked
	bit	integer	defines bit position if data exist in bitfield with other data
adjustmentSlider	stepUpMessage	string	message to send on step up button
	stepDownMessage	string	message to send on step down button
radioGroup	radioButtons	array of valuedradiobutton objects	
valuedRadioButton	isCombo	boolean	defines if combo is associated with
	setValue	string	value to send radiobutton selected
	values	array of comboBox:list objects	used for combo if isCombo is true
checkBoxGroup	bits	array of integers	defines bit positions in bitfield for each checkbox in group

UML-toimintakaavio JSON-jäsentimestä



UML-luokkakaavio DynCommander-sovelluksesta